HW2 ME333 Winter 2025

Author: Zhengyang Kris Weng

18. Invoking the gcc compiler with a command like gcc myprog.c -o myprog actually initiates four steps. What are the four steps called, and what is the output of each step?

The four steps are preprocessing, compiling, assemblying, and linking:
Preprocessing:

```
Processes directives like #include, #define, #ifdef, etc.
Output: A translation unit that includes expanded macros, included files,
and removed comments.
```

Compiling:

```
Converts the preprocessed code into assembly code, checking for syntax and
semantic errors.
Output: Assembly code (.s or .asm file), which is platform-specific but not
yet machine code.
```

Assembling:

```
Translates assembly code into machine code (object code), which is in
binary format but not fully linked.
Output: Object file (.o or .obj), containing machine code and unresolved
references to external symbols.
```

Linking:

```
Combines object files and libraries into a single executable or shared
library, resolving symbol references.
Output: Executable file (e.g., .exe, .out, or binary) or shared library
(e.g., .so, .dll), ready to run.
```

19. What is main ’s return type, and what is the meaning of its return value?

Int. The return value is used to indicate the status of the program when it finishes. A return value of 0 generally indicates successful execution.

A non-zero return value typically represents an error, where the value often serves as an error code to provide more specific information about the failure. Different values may correspond to different types of

errors depending on the program's convention.

21.Consider three unsigned chars, i , j , and k , with values 60, 80, and 200, respectively. Let sum also be an unsigned char. For each of the following, give the value of sum after performing the addition. (a) sum = i+j; (b) sum = i+k; (c) sum = j+k

a. i+j = 60 + 80 = **140**
b. i+k = 60 + 200 = 260, 260%256 = **4**
c. j+k = 80 + 200 = 280, 280%256 = **24**

22. For the variables defined as int a=2, b=3, c; float d=1.0, e=3.5, f; give the values of the following expressions. (a) f = a/b; (b) f = ((float) a)/b; (c) f = (float) (a/b); (d) c = e/d; (e) c = (int) (e/d); (f) f = ((int) e)/d;

a. a/b = **0.0**
b. ((float) a) / b = **0.6667**
c. (float) (a / b) = **0.0**
d. e / d = **3**
e. (int) (e / d) = **3**
f. ((int) e) / d = **3.0**

27. You have written a large program with many functions. Your program compiles without errors, but when you run the program with input for which you know the correct output, you discover that your program returns the wrong result. What do you do next? Describe your systematic strategy for debugging.

For debugging a large program, I would try to:

1. Reproduce the error with specific input.
2. Add print statements to trace variable values.
3. Test each function with simple inputs.
4. Step through the program using a debugger.
5. Isolate the faulty section of code and fix it.
6. Rerun the program with the original input to confirm the fix.

28. Erase all the comments in invest.c , recompile, and run the program to make sure it still functions correctly. You should be able to recognize what is a comment and what is not. Turn in your modified invest.c code.

Please see `invest_modified.c` attached.

30. Consider this array definition and initialization: int x[4] = {4, 3, 2, 1}; For each of the following, give the value or write “error/unknown” if the compiler will generate an error or the value is unknown. (a) x[1] (b) *x (c) *(x+2) (d) (*x)+2 (e) *x[3] (f) x[4] (g) *(&(x[1]) + 1)

a. x[1] = 3
b. *x = 4
c. *(x+2) = 2
d. *(x) + 2 = 6

e *x[3] = error/unknown
f. x[4] = error/unknown
g. *(&(x[1])+1) = 2

31. For the (strange) code below, what is the final value of i ? Explain why. int i,k=6; i = 3* (5>1) + (k=2) + (k==6);

For the 3 subcomponents of this operation, we have: (5>1) = True (1); (k = 2) = 2; (k==6) = False (0);

So this gives us:
i = 3 * 1 + 2 + 0 = 5

32. As the code below is executed, give the value of c in hex at the seven break points indicated, (a)-(g).

unsigned char a=0x0D, b=0x03, c; c = ~a; // (a) c = a & b; // (b) c = a | b; // (c) c = a ^ b; // (d) c = a >> 3; // (e) c = a << 3; // (f) c &= b; // (g)

a = 0x0D = 0000 1101
b = 0x03 = 0000 0011

a. c = ~a = 1111 0010 = **0xF2**
b. c = a & b = 0000 0001 = **0x01**
c. c = a | b = 0000 1111 = **0x0F**
d. c = a ^ b = 0000 1110 = **0x0E**
e. c = a >> 3 = 0000 0001 = **0x01**
f. c = a << 3 = 0110 1000 = **0x68**
g. c &= b = 0000 0000 = **0x00**

34. Write a program to generate the ASCII table for values 33 to 127. The output should be two columns: the left side with the number and the right side with the corresponding character. Turn in your code and the output of the program.

Please see `ascii_converter.c` for more details. Sample output:

```
ASCII Table (33 to 127)
------------------------
Number  Character
------------------------
33      !
34      "
35      #
36      $
37      %
38      &
39      '
40      (
41      )
42      *
43      +
```

```
44      ,
45      -
46      .
47      /
48      0
49      1
50      2
51      3
52      4
53      5
54      6
55      7
56      8
57      9
58      :
59      ;
60      <
61      =
62      >
63      ?
64      @
65      A
66      B
67      C
68      D
69      E
70      F
71      G
72      H
73      I
74      J
75      K
76      L
77      M
78      N
79      O
80      P
81      Q
82      R
83      S
84      T
85      U
86      V
87      W
88      X
89      Y
90      Z
91      [
92      \
93      ]
94      ^
95      _
96      `
97      a
```

```
98      b
99      c
100     d
101     e
102     f
103     g
104     h
105     i
106     j
107     k
108     l
109     m
110     n
111     o
112     p
113     q
114     r
115     s
116     t
117     u
118     v
119     w
120     x
121     y
122     z
123     {
124     |
125     }
126     ~
127
```

35. We will write a simple bubble sort program to sort a string of text in ascending order according to the ASCII table values of the characters.

Please see `bubble_sort.c` attached for more details.